CHAPTER 2

PHP and Objects

Objects were not always a key part of the PHP project. In fact, they have been described as an afterthought by PHP's designers.

As afterthoughts go, this one has proved remarkably resilient. In this chapter, I introduce coverage of objects by summarizing the development of PHP's object-oriented features.

We will look at

- *PHP/FI 2.0*: PHP, but not as we know it.
- *PHP 3*: Objects make their first appearance.
- *PHP 4*: Object-oriented programming grows up.
- *PHP 5*: Objects at the heart of the language.
- *PHP 6*: A glimpse of the future

The Accidental Success of PHP Objects

With so many object-oriented PHP libraries and applications in circulation, to say nothing of PHP 5's extensive object enhancements, the rise of the object in PHP may seem like the culmination of a natural and inevitable process. In fact, nothing could be further from the truth.

In the Beginning: PHP/FI

The genesis of PHP as we know it today lies with two tools developed by Rasmus Lerdorf using Perl. PHP stood for Personal Homepage Tools. FI stood for Form Interpreter. Together, they comprised macros for sending SQL statements to databases, processing forms, and flow control.

These tools were rewritten in C and combined under the name PHP/FI 2.0. The language at this stage looked different from the syntax we recognize today, but not *that* different. There was support for variables, associative arrays, and functions. Objects, though, were not even on the horizon.

Syntactic Sugar: PHP 3

In fact, even as PHP 3 was in the planning stage, objects were off the agenda. As today, the principal architects of PHP 3 were Zeev Suraski and Andi Gutmans. PHP 3 was a complete rewrite of PHP/FI 2.0, but objects were not deemed a necessary part of the new syntax.

According to Zeev Suraski, support for classes was added almost as an afterthought (on 27 August 1997, to be precise). Classes and objects were actually just another way to define and access associative arrays.

Of course, the addition of methods and inheritance made classes much more than glorified associative arrays, but there were still severe limitations as to what you could do with your classes. In particular, you could not access a parent class's overridden methods (don't worry if you don't know what this means yet; I will explain later). Another disadvantage that I will examine in the next section was the less than optimal way that objects were passed around in PHP scripts.

That objects were a marginal issue at this time is underlined by their lack of prominence in official documentation. The manual devoted one sentence and a code example to objects. The example did not illustrate inheritance or properties.

PHP 4 and the Quiet Revolution

If PHP 4 was yet another ground-breaking step for the language, most of the core changes took place beneath the surface. The Zend Engine (its name derived from **Ze**ev and A**nd**i) was written from scratch to power the language. The Zend Engine is one of the main components that drive PHP. Any PHP function you might care to call is in fact part of the high level extensions layer. These do the busy work they were named for, like talking to database APIs or juggling strings for you. Beneath that the Zend Engine manages memory, delegates control to other components, and translates the familiar PHP syntax you work with every day into runnable bytecode. It is the Zend Engine we have to thank for core language features like classes.

From our *object*ive perspective, the fact that PHP 4 made it possible to override parent methods and access them from child classes was a major benefit.

A major drawback remained, however. Assigning an object to a variable, passing it to a function, or returning it from a method, resulted in a copy being made. So an assignment like this

```
$my_obj = new User('bob');
$other = $my_obj;
```

resulted in the existence of two `User` objects, rather than two references to the same `User` object. In most object-oriented languages you would expect assignment by reference, rather than by value as here. This means that you pass and assign handles that point to objects rather than copy the objects themselves. The default pass-by-value behavior resulted in many obscure bugs as programmers unwittingly modified objects in one part of a script, expecting the changes to be seen via references elsewhere. Throughout this book, you will see many examples in which I maintain multiple references to the same object.

Luckily, there was a way of enforcing pass-by-reference, but it meant remembering to use a clumsy construction.

Assign by reference as follows:

```
$other =& $my_obj;
// $other and $my_obj point to same object
```

Pass by reference as follows:

```
function setSchool( & $school ) {
    // $school is now a reference to not a copy of passed object
}
```

And return by reference as follows:

```
function & getSchool( ) {
    // returning a reference not a copy
    return $this->school;
}
```

Although this worked fine, it was easy to forget to add the ampersand, and it was all too easy for bugs to creep into object-oriented code. These were particularly hard to track down, because they rarely caused any reported errors, just plausible but broken behavior.

Coverage of syntax in general, and objects in particular, was extended in the PHP manual, and object-oriented coding began to bubble up to the mainstream. Objects in PHP were not uncontroversial (then, as now, no doubt), and threads like "Do I need objects?" were common flame-bait in mailing lists. Indeed, the Zend site played host to articles that encouraged object-oriented programming side by side with others that sounded a warning note.

Pass-by-reference issues and controversy notwithstanding, many coders just got on and peppered their code with ampersand characters. Object-oriented PHP grew in popularity. As Zeev Suraski wrote in an article for DevX.com (`http://www.devx.com/webdev/Article/10007/0/page/1`):

> *One of the biggest twists in PHP's history was that despite the very limited functionality, and despite a host of problems and limitations, object-oriented programming in PHP thrived and became the most popular paradigm for the growing numbers of off-the-shelf PHP applications. This trend, which was mostly unexpected, caught PHP in a suboptimal situation. It became apparent that objects were not behaving like objects in other OO languages, and were instead behaving like [associative] arrays.*

As noted in the previous chapter, interest in object-oriented design became obvious in sites and articles online. PHP's official software repository, PEAR, itself embraced object-oriented programming. Some of the best examples of deployed object-oriented design patterns are to be found in the packages that PEAR makes available to extend PHP's functionality.

With hindsight, it's easy to think of PHP's adoption of object-oriented support as a reluctant capitulation to an inevitable force. It's important to remember that, although object-oriented programming has been around since the sixties, it really gained ground in the mid-nineties. Java, the great popularizer, was not released until 1995. A superset of C, a procedural language, C++ has been around since 1979. After a long evolution, it arguably made the leap to the big time during the nineties. Perl 5 was released in 1994, another revolution within a formerly procedural language that made it possible for its users to think in objects (although some argue that Perl's object-oriented support still feels like something of an afterthought). For a small procedural language, PHP developed its object support remarkably fast, showing a real responsiveness to the requirements of its users.

Change Embraced: PHP 5

PHP 5 represented an explicit endorsement of objects and object-oriented programming. That is not to say that objects are now the only way to work with PHP (this book does not say that either, by the way). Objects, are, however, now recognized as a powerful and important means for developing enterprise systems, and PHP fully supports them in its core design.

Objects have moved from afterthought to language driver. Perhaps the most important change is the default pass-by-reference behavior in place of the evils of object copying. This is only the beginning though. Throughout this book, and particularly this part of it, we will encounter many more changes that extend and enhance PHP's object support, including argument hinting, private and protected methods and properties, the `static` keyword, namespaces, and exceptions, among many others.

PHP remains a language that supports object-oriented development, rather than an object-oriented language. Its support for objects, however, is now well enough developed to justify books like this one that concentrate on design from an exclusively object-oriented point of view.

Into the Future

As I write this, PHP 6 is still some way off, but it is under active development. It will be built on an entirely new generation of the Zend Engine (ZE3), and will provide built-in support for Unicode sting handling, which will make the language better able to support internationalization. This means you will be able to use all PHP's string functions without worrying about whether they can work with the current character set. In the past, developers had to use multibyte equivalents for many common functions—a frustrating and error-prone task. As internationalization becomes more and more important, this core feature is fast becoming essential in any serious programming language..
In some ways the future is already here. A feature that was slated for PHP 6 has now found its way into PHP 5 (as of PHP 5.3): support for namespaces. Namespaces let you create a naming scope for classes and functions so that you are less likely to run into duplicate names as you include libraries and expand your system. They also rescue you from ugly but necessary naming conventions like this:

```
class megaquiz_util_Conf {
}
```

Class names like this are one way of preventing clashes between packages, but they can make for tortuous code.

At the time of this writing, it looks like support for hinted return types is once again slated for PHP 6. This will allow you to declare in a method or function's declaration the object type it returns. This commitment will then be enforced by the PHP engine. Hinted return types will further improve PHP's support for pattern principles (principles such as "code to an interface, not an implementation"). I hope to revise this book to cover that feature!

Advocacy and Agnosticism: The Object Debate

Objects and object-oriented design seem to stir passions on both sides of the enthusiasm divide. Many excellent programmers have produced excellent code for years without using objects, and PHP continues to be a superb platform for procedural web programming.

This book naturally displays an object-oriented bias throughout, a bias that reflects my object-infected outlook. Because this book *is* a celebration of objects, and an introduction to object-oriented design, it is inevitable that the emphasis is unashamedly object oriented. Nothing in this book is intended, however, to suggest that objects are the one true path to coding success with PHP.

As you read, it is worth bearing in mind the famous Perl motto, "There's more than one way to do it." This is especially true of smaller scripts, where quickly getting a working example up and running is more important than building a structure that will scale well into a larger system (scratch projects of this sort are known as "spikes" in the eXtreme Programming world).

Code is a flexible medium. The trick is to know when your quick proof of concept is becoming the root of a larger development, and to call a halt before your design decisions are made for you by sheer weight of code. Now that you have decided to take a design-oriented approach to your growing project, there are plenty of books that will provide examples of procedural design for many different kinds of projects. This book offers some thoughts about designing with objects. I hope that it provides a valuable starting point.

Summary

This short chapter placed objects in their context in the PHP language. The future for PHP is very much bound up with object-oriented design. In the next few chapters, I take a snapshot of PHP's current support for object features, and introduce some design issues.